A generic library of problem solving methods for scheduling applications

A Generic Library of Problem Solving Methods for Scheduling Applications

Dnyanesh G. Rajpathak, Enrico Motta, Zdenek Zdrahal, and Rajkumar Roy

Abstract—In this paper, we propose a generic library of problem-solving methods for scheduling applications. Although some attempts have been made in the past at developing the libraries of scheduling problem-solvers, these only provide limited coverage. Many lack generality, as they subscribe to a particular scheduling domain. Others simply implement a particular problem-solving technique, which may be applicable only to a subset of the space of scheduling problems. In addition, most of these libraries fail to provide the required degree of depth and precision. In our approach, we subscribe to the Task-Method-Domain-Application knowledge modeling framework which provides a structured organization for the different components of the library. At the task level, we construct a generic scheduling task ontology to formalize the space of scheduling problems. At the method level, we construct a generic problem-solving model of scheduling that generalizes from the variety of approaches to scheduling problem-solving, which can be found in the literature. The generic nature of this model is demonstrated by constructing seven methods for scheduling as an alternative specialization of the model. Finally, we validated our library on a number of applications to demonstrate its generic nature and effective support for developing scheduling applications.

Index Terms—Knowledge modeling, knowledge engineering, knowledge-based systems, task-method-domain-application modeling, ontologies, problem solving methods, scheduling.

1 Introduction

Scheduling [3] is the central theme of this paper. As a first-level approximation, we can say that scheduling deals with *the assignment of jobs and activities to resources and time ranges in accordance with relevant constraints and requirements*. Typical domains of scheduling include manufacturing scheduling, project scheduling, resource allocation, transportation scheduling, mass transit scheduling, scheduling nurse shifts in hospital, and air gate assignment. Each scheduling domain imposes its unique constraints and requirements, which need to be obeyed by a scheduler while devising a schedule because they determine the space of a valid solution. A process of constructing a valid schedule becomes even more challenging due to the *uncertain*, *dynamic*, and *unpredictable* circumstances [24] that occur in an environment where the scheduling task has to take place [8], [23]. For instance, rush orders arrive without prior notice, the existing resources become unavailable and, moreover, cost criteria also plays a crucial role when multiple solutions can be admissible for a particular problem, and some of them are "better" than others.

One of the earliest research initiatives in scheduling, Operations Research (OR) [62], aimed at finding an optimal solution, but optimization normally suffers from combinatorial complexity that can be proven NP-hard [25], [26]. Generally speaking, the OR techniques [27] are restricted to rigid and static models with limited expressive power and when implemented to solve real-life scheduling problems, their sophisticated mathematical algorithms result in intractability, mainly because the problem space of the real-life scheduling problems is normally ill-structured [28], [57]. In order to overcome the limitations observed in OR techniques [62], several knowledge-based techniques from artificial intelligence (AI) have been used to solve the scheduling problem and, consequently, various intelligent scheduling systems [9], [13], [21], [22], [32], [45], [52], [63], [64], [65] have been developed over the last 20-year period.[1] Although these systems have used various AI techniques successfully, they were hardwired in nature and the domain specific nature of these systems restricted their reusability within a single domain. As pointed out by Kruger [37] and Neches et al. [49], monolithic systems are difficult to maintain because these systems need to be constructed from scratch every time the nature of domain changes.

Reusability is the main concern of research in knowledge modeling. Here, the construction of a knowledge-based system (KBS) can be realized by applying libraries of problem-solving methods (PSMs) [5], [6], [46], [73], [75]. An *ontology* [29] and a *PSM* [30] are the two central components in library construction mainly because they enhance knowledge *sharing* and *reusability* over wider domains. An *ontology* [29] can be seen as an information model that explicitly describes the various entities and abstractions that exist in a universe of discourse, along with their properties. Much work on reusable components for knowledge-based systems [5], [6], [46], [73], [75] relies on ontologies to specify formally generic classes of knowledge-intensive tasks, e.g., design and planning, as well as the ontological commitments associated with

- *D.G. Rajpathak, E. Motta, and Z. Zdrahal are with the Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK. E-mail: {d.g.rajpathak, e.motta, z.zdrahal}@open.ac.uk.*
- *R. Roy is with the School of Industrail and Manufacturing Science, Cranfield University, Cranfield, Beford, MK43 0AL, UK. E-mail: r.roy@cranfield.ac.uk.*

Manuscript received 9 Aug. 2005; revised 27 Jan. 2006; accepted 15 Feb. 2006; published online 20 Apr. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0338-0805.

1. An excellent review of the intelligent scheduling systems can be found in [54].

knowledge-intensive problem-solvers. In the former case, we use the term *task ontologies*, in the latter *method ontologies*. The task ontology [44], [46] is method, domain, and application independent, while the method ontology [12] provides a vocabulary necessary to characterize the problem solving behavior of a generic task. A PSM is a domain independent specification of the reasoning process underlying a KBS. As discussed by van Heijst et al. [74], a PSM can be used as a model-based template to guide the knowledge acquisition process and it can also be used to develop robust and maintainable applications by reuse, e.g., [39], [46], [58]. A PSM can either be *task specific* or *task independent*. Task specific PSMs are developed to tackle the specific types of Generic Tasks [10], [11], such as planning [73], parametric design [46], [75], and they consist of the following components [5], [18]: 1) *Functional specification* is a declarative description of input/output behavior stating what can be achieved by a PSM, 2) *requirements* state the domain knowledge required by a PSM to achieve its functionality, and 3) *operational specification* is a reasoning process consisting of inference structure, the knowledge, and the control flow, and it ensures that if the correct knowledge is provided, then the functional specification of a PSM can be achieved successfully. Task independent PSMs on the other hand provide high-level reasoning steps in terms of a generic paradigm, e.g., search [50].

Some attempts have been made in the past at developing libraries of scheduling PSMs [35], [38], [69], [72], but these proposals provide limited results. In some cases [35], these libraries subscribe to specific scheduling domains, which limit their reusability to these domains. In other cases [38], they subscribe to a specific "problem-solving technique," e.g., constraint satisfaction, which provides limited "conceptual leverage" to tease out all the knowledge-intensive activities that take place in scheduling and also provides limited support for knowledge acquisition. Moreover, none of the existing libraries provide comprehensive coverage of the different knowledge-intensive PSMs, e.g., Propose & Improve [46] and Propose & Exchange [53]. For instance, the CommonKADS library [69] is only comprised of the Propose & Revise method [40], [41]. The partial coverage exhibited by these libraries fails to reason about the different types of schedules, such as completion, constraint and requirement violation, and optimization.

The primary aim of our work is to develop a task-specific, but domain independent, library of scheduling PSMs to overcome all the shortcomings observed in the existing proposals. Our library is task-specific because it is developed to solve the scheduling task, but it is domain independent because it does not subscribe to any specific application domain of scheduling. In our approach, we subscribe to the Task-Method-Domain-Application (TMDA) knowledge modeling framework [46], which provides a structured way to organize the components of our library. Our library is formalized by using the Operational Conceptual Modelling Language (OCML) [46], which provides support for producing sophisticated specifications, as well as mechanisms for operationalizing definitions to provide a concrete reusable resource to support knowledge acquisition and system development. Import/export mechanisms from OCML to standards, such as OWL [43] and Ontolingua [17] ensure symbol-level interoperability. The TMDA framework and the OCML language are discussed in the next section.

The rest of the paper is organized as follows: In the following section, we introduce the TMDA knowledge modeling framework and also provide an overview of OCML. In Section 3, we discuss a generic scheduling task ontology. In Section 4, we describe the construction of generic problem-solving model of scheduling (henceforth `Generic-Schedule`). In Section 5, we describe the engineering of the Propose & Revise method [41] to demonstrate how alternative PSMs can be constructed simply by reusing and specializing `Generic-Schedule`. The evaluation of the library is discussed in Section 6. In Section 7, we compare our work with other alternative proposals in the field. Finally, in Section 8, we conclude the paper by summarizing the main contributions of this work.

2 Task-Method-Domain-Application Knolwedge Modeling Framework and OCML

2.1 Task-Method-Domain-Application Knowledge Modeling Framework

Although various knowledge modeling frameworks, such as Generic Tasks Structures [10], Role-Limiting Methods [41], [42], Protégé-II [48], CommonKADS [61], MIKE [2], Components of Expertise [67], EXPECT [70], and GDM [71], have been proposed in the knowledge modeling community to provide a structured organization for the library of problem-solving components, our library subscribes to the TMDA framework [46]. Some of the key reasons for subscribing to the TMDA knowledge modeling framework are as follows: 1) TMDA uses different kinds of formal ontologies to specify the generic structure of a class of problems (task ontology) and the knowledge requirements of PSMs (method ontologies), 2) similarly with CommonKADS and Components of Expertise, the TMDA framework introduces a systematic separation between the task, method, and domain components, but it extends this partition by introducing an "application" component. The application component clearly distinguishes between a mapping knowledge and application-specific knowledge. The mapping knowledge is used to interpret the task and method components with multidimensional domain models and it is also associated with the domain independence of PSMs, and 3) although Protégé-II [48] introduces a notion of application ontology to formalize the application-specific knowledge, in their approach, the construction of application ontology is more of a creative process with very limited support for explaining the actual content of application ontology itself [31].

As shown in Fig. 1, in compliance with the TMDA framework, our library organization can be seen as a four tier hierarchy. At the task level, we formalize the scheduling task by developing its generic task ontology. This scheduling task ontology is mapped at the method level, where we first develop `Generic-Schedule` that generalizes from the variety of approaches to scheduling problem-solving, and

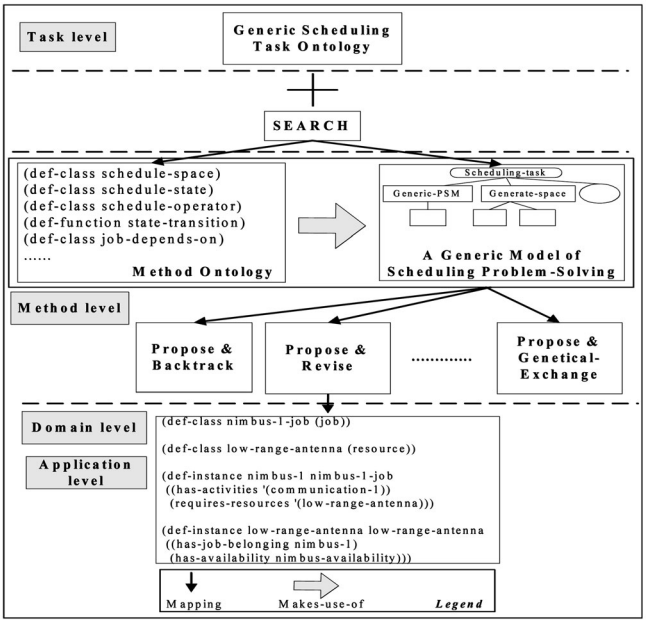


Fig. 1. Library organization in compliance with the Task-Method-Domain-Application framework.

then seven different PSMs are defined by reusing `Generic-Schedule`. Finally, these PSMs are applied to solve the scheduling application from different domains.

2.2 Operational Conceptual Modeling Language

The OCML knowledge modeling language was originally developed to provide an operational modeling capability for the VITAL workbench [16]. Although OCML can be used to support different knowledge modeling approaches, it is mainly developed to provide a modeling support for the TMDA framework.

OCML supports knowledge-level modeling specification by supporting the following primitives: classes, instances, relations, functions, rules, procedures, and axioms. OCML supports the specification of classes and instances and the inheritance of slots and values in terms of an *is-a* hierarchy. The classes and instances in OCML are represented by the Lisp macro `def-class` and `def-instance`, respectively. The instances are the members of a class, which takes as arguments the name of the instance, the most specific class to which the instance belongs, optional documentation, and a number of slot-value pairs. In OCML, relations are specified by the Lisp macro called `def-relation`. OCML relations allow the users to define labeled n-ary relationships between different entities and they take as arguments the name of a relation, its argument schema, an optional documentation, and a number of relation options. OCML functions are specified by the Lisp macro `def-function`. The functions take as arguments the name of a function, its argument list, an optional variable indicating the output in general, and a mapping between a list of input arguments and its output argument. OCML also supports specification of backward and forward rules. In OCML, the *functional terms* are represented as a constant, a variable, a string, or they can be constructed by special term constructor, e.g., `if`, `cond`, `the`, `setofall`, `findall`, and `in-environment`. The *control terms* are represented to model the problem-solving actions and their sequence of execution, e.g., `repeat`, `loop`, `if`, and `cond`. The *logical expressions* are mechanisms to specify the logical expressions, e.g., `and`, `or`, `not`, `forall exists`.

3 A Generic Task Ontology of the Scheduling Task

Our scheduling task ontology formalizes the space of scheduling tasks without subscribing to any specific application domain of scheduling or the way a scheduling task can be solved. This task ontology subscribes to the job centric viewpoint [7] as opposed to the resource centric viewpoint [7] and it relies on two underlying ontologies, Base Ontology and Simple Time ontology. Base Ontology provides the definitions for basic modeling concepts, such as frames, classes, slots, relations, functions, roles, numbers, list, and sets. The Simple Time ontology makes use of Allen's [1] representation of standard time and relations. Earlier analyses of the scheduling task ontology can be found in [47]. The OCML version of the ontology can be browsed using the WebOnto environment at: http://plainmoor.open.ac.uk:3000/webonto.

3.1 A Generic Specification of the Scheduling Task

In our framework, the scheduling task is formally represented as a mapping from a nine-dimensional space: $\{\mathrm{J}, \mathrm{A}, \mathrm{R}, \mathrm{Tr}, \mathrm{C}, \mathrm{Req}, \mathrm{Pr}, \mathrm{Cf}, \mathrm{Cr}\}$ to a schedule, S. They are described below:

- *Job*, $\mathbf{J} = \{\mathrm{j}_1, \ldots, \mathrm{j}_\mathrm{m}\}$. A set of jobs to be assigned to a set of resources for their execution.
- *Activities*, **A**. For each job, j_m, there are A_m uniquely associated activities which are denoted as $\mathrm{A}_\mathrm{m} = \{\mathrm{a}_{\mathrm{m}1}, \ldots, \mathrm{a}_{\mathrm{m}n}\}$.
- *Resources*, $\mathbf{R} = \{\mathrm{r}_1, \ldots, \mathrm{r}_\mathrm{p}\}$. A set of resources to which the jobs and activities can be assigned for their execution.
- *Constraints*, $\mathbf{C} = \{\mathrm{c}_1, \ldots, \mathrm{c}_\mathrm{n}\}$. A set of constraints that must not be violated by a schedule.
- *Requirements*, $\mathbf{Req} = \{\mathrm{req}_1, \ldots, \mathrm{req}_\mathrm{k}\}$. A set of requirements that describe the desired properties of a solution schedule.
- *Schedule time range*, **Tr**. The time horizon in which the schedule takes place. It is represented by a start and an end time.
- *Preferences*, $\mathbf{P} = \{\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{t}\}$. A set of criteria for choosing among competing solution schedules. Each preference defines a partial order over the set of solution schedules.
- *Cost function*, **Cf**. A function, which computes the cost of a solution schedule.
- *Solution criterion*, **Cr**. A mapping from a schedule S to $\{\mathrm{True}, \mathrm{False}\}$, which determines whether a candidate schedule is a solution, which requires S to be *correct*, *complete*, *consistent*, and *feasible*—see below for the definitions of these properties. More restrictive criteria may specify an optimality condition on a solution schedule, but, due to the unique specification of the optimality criterion in different scheduling domains, we do not specify

the default optimization criterion, but specialize it according to the specific application.

- *Schedule*, $\mathbf{S} = \{s_1, \ldots, s_w\}$. A schedule is a set of quadruples of the form, $\{< j_m, a_{mn}, r_k, jtr_{m,n,k} >\}$, where j_m is a job, a_{mn} is an activity associated with j_m, r_k is a resource, and $jtr_{m,n,k}$ is the *job time range* associated with the assignment of j_m and a_{mn} to resource r_k. The job time range is represented in terms of the earliest and latest start and end times. It is a subinterval of **Tr**.

Below, we define the schedule validation criterion used to check the validity of a schedule:

- S is *correct* if, for every job j_m and activity a_{mn}, the pair j_m a_{mn} appears no more than once in S. This criterion makes sure that each job has a unique association with only one schedule quadruple.
- S is *complete* if, for each activity a_{mn} in A_m, there exists a quadruple q in S such that $q =< j_m, a_{mn}, r_k, jtr_{m,n,k} >$.
- S is *consistent*, if it does not violate any applicable constraints in **C**.
- S is *feasible* if it satisfies all the requirements in **Req**.
- S is *optimal* if it is a solution schedule and no other solution schedule has a lower cost than S.

3.2 Scheduling Task and Default Schedule Solution

Our task modeling framework characterizes a generic task in terms of input and output roles, preconditions, and a goal expression [18], [19], [20]. The nine-dimensional space $\{J, A, R, Tr, C, Req, Pr, Cf, Cr\}$ described in the previous section provides the input roles to formalize the scheduling task and the output role of the scheduling task is to generate a schedule, S. The precondition imposed on the scheduling task states that jobs and resources are required for a meaningful specification of the scheduling task, while the goal expression states that the schedule validation criterion (see Section 3.1) must hold for the output schedule to become a solution.

3.3 Modeling the Notion of Job and Resource

The class `job` represents an entity that has a list of activities and can be assigned over available resources and time ranges for its execution. The class `job` has the following attributes (represented by the slots in OCML): **Requires-Resource** specifies a number of resources on which a job can be assigned for its completion. The minimum-cardinality restriction imposed on this slot is 1, which states that at least one resource must be assigned to a job for the successful completion of a job. **Requires-Resource-Type** specifies the particular type of resources that are needed to carry out a job, e.g., a machine type. **Has-Activities** state that every job can have a list of activities that need to be performed to accomplish a job. **Has-Time-Range** represents a time range within which a job must complete its execution. This slot inherits the values of the class `job-time-range` (see Section 3.5). **Has-Load** represents the number of resources required by a job which are represented by the attribute **requires-resource**. **Has-Due-Date** represents the calendar date by which a job must be dispatched to a customer. **Has-Duration** represents the total amount of time that has elapsed between the earliest and the latest start and end time of a job time range.

The class `resource` is a finite supply entity on which jobs can be assigned for their completion. The class `resource` has the following attributes: **Handles-Job** represents the specific jobs each resource can handle for its execution, e.g., job_1. **Handles-Activity** represents the activities each resource is capable of handling. **Has-Availability** represents the time interval during which a resource is available to accomplish jobs and jobs must maintain the resource availability period. The resource availability period can take multiple time intervals, which allows us to handle a situation when a particular resource becomes unavailable after a certain period and becomes available again. **Has-Capacity** represents the maximum number of jobs each resource can handle at any given time in a schedule. The resource capacity is represented as an *integer*.

In scheduling, any two jobs that share the same unary resource may generate a conflicting situation if the time ranges of these two jobs overlap. To avoid such inconsistency, we define an axiom named `resource-capacity`, which states that, for a given unary capacitated resource "r_i" with capacity "n" in schedule "S," there should not exist two jobs, j_i and j_k, such that j_i and j_k require r_i and the time ranges of j_i and j_k overlap with each other.

3.4 Modeling Constraints and Requirements

In our task ontology, we distinguish between constraints and requirements, even though existing approaches [34], [44], [66] fail to make such a distinction. The class `constraint` defines a property that must not be violated by a *consistent solution*. For instance, the "resource capacity constraint" in the weekly ship-maintenance application (see Section 5.1) restricts the maximum number of ship-maintenance jobs each ship-maintenance resource can handle. The class `requirement` specifies a property that a *feasible solution* has to satisfy. For instance, the "job priority requirement" in the weekly ship-maintenance application (see Section 6) states that a ship-maintenance job with higher duration gets a priority over other competing jobs for its assignment. In our model, we do not differentiate between hard and soft constraints mainly because soft constraints are neither prescriptive nor proscriptive, but, in reality, a solution schedule that satisfies a maximum number of soft constraints is treated as a better solution [15], [60]. In our model, we use the notion of cost and preferences in order to determine the quality of a solution schedule.

3.5 Representing the Time Ranges

The class `job-time-range` represents the period in which a job or activity can be executed and it has the following attributes: **Has-Earliest-Start-Time** represents the earliest time a particular job can start its execution. **Has-Latest-Start-Time** represents the latest time a particular job must start. **Has-Earliest-End-Time** represents the earliest time a particular job can finish. **Has-Latest-End-Time** represents the latest time a particular job must finish. **Has-unit-of-time** simply represents the unit used to specify the time, e.g., second, minute, and hour.

The class time-range represents a schedule horizon and a resource availability period and it has the following attributes: **Has-Start-Time** is the time by which a task must start. **Has-End-Time** is the time by which a task must end. **Has-Unit-of-Time** is the unit in which the time is specified.

3.6 Representing Cost, Cost Function, and Preference

The scheduling task not only deals with the satisfaction of constraints or maintenance of requirements, but it can also be seen as a combinatorial optimization problem [36], where the evaluation function, e.g., minimization of cost or maximization of resource utilization, needs to be optimized. Our task ontology provides the two classes that allow us to capture the knowledge needed to rank solutions: preference and cost-function. The class preference allows us to describe task knowledge needed to assess whether a solution can be regarded as better than another. Once the relevant preferences are acquired, we use the class cost-function to develop an optimization criterion for a given scheduling problem. As preferences tend to be heterogeneous and have different costs associated with them, it is crucial to take into account these preferences while calculating the cost of a schedule. Therefore, it is important to emphasize that a cost function in our model may not necessarily be numeric and often some non-Archimedean criterion [46] may be applied as well. Our task ontology models preferences as binary relations, which define a partial order over schedules. The class cost-function is defined as a mapping from schedules to costs. A cost is modeled either as a real-number or as an n-dimensional vector. The role of the class cost-function is to define a single optimization criterion, which is both consistent with and subsumes the various criteria expressed by the various preferences. In our task ontology, these conditions are specified by two axioms: 1) cost-subsumes-preferences and 2) cost-preference-consistency http://plainmoor.open.ac.uk:3000/webonto. The first axiom states that the cost-function should enforce the partial order expressed by any relevant preference. The second axiom states that the cost function should not violate any preference. Both the axioms make use of the association between a cost-function and a cost-order relation, which expresses the partial order defined by the cost function.

3.7 Representing a Schedule

The class schedule represents the actual mapping of a job and its activities to resources within a time range and it is represented in terms of a set of job-assignment quadruples. The class job-assignment models a quadruple of the form <job activity resource job-time-range>.

4 A Generic Model of Scheduling Problem Solving

Generic-Schedule takes as an input the scheduling task ontology and it subscribes to *search* [50] as a problem-solving technique. Generic-Schedule subscribes to a top-down approach to schedule construction, whereby the top-level scheduling task from the task ontology is decomposed into a finite number of (sub)tasks and (sub)methods are proposed to solve these subtasks. This decomposition allows us to construct the new PSMs simply by reusing or specializing the tasks and methods from Generic-Schedule. A more detailed discussion of Generic-Schedule and other components of the library can be found in [55], [56]. The OCML version of Generic-Schedule can be browsed using the WebOnto environment at: http://plainmoor.open.ac.uk:3000/webonto.

4.1 A Generic Method Ontology

A generic method ontology [12], [46] provides a vocabulary necessary to characterize the search-based problem solving behavior of Generic-Schedule.

4.1.1 Schedule Space, Schedule State, and State Transition

The space of scheduling problem-solving can be represented by means of a *state-space* and *operators* [56]. The class schedule-space indicates a problem space associated with the scheduling task and it is composed of a set of schedule states. Each schedule state associated with a schedule space is represented by the class schedule-state and each schedule state has a unique association with a schedule, say S_{sch}, from the task ontology. In an initial or root state, a schedule is incomplete because all the jobs are still unassigned, while, in the solution state, it satisfies all the default solution criteria, defined in the scheduling task ontology. The relation state-transition enables a scheduling agent to transit from an initial state to the solution state.

4.1.2 Schedule Operators and Job Dependency Network

Each schedule operator extends a partial schedule state by assigning jobs to resources and time ranges. The schedule-extension-resource-operator is used to assign jobs to the correct resources. The schedule-extension-time-range-operator is used to assign jobs to their time ranges. Both the operators have the attribute called applicable-to-jobs, which is instantiated with the application specific knowledge to model the jobs that can be assigned by application of the relevant operators. Finally, the relation schedule-operator-order determines the order in which the resource and the time range operators can be applied to achieve the job assignments.

The job dependency network [21] in the method ontology makes the scheduling problem-solving process more of a "tightly coupled" one because it reflects the job assignment effect of a particular job on other unassigned jobs. Following are the key relations and function from the job dependency network: **Job-Depends-On** states that the assignment of a job, j_1, depends on another job, j_2. **Job-Affects** is the inverse of the **Job-Depends-On** relation. **Job-Assignable** is a binary relation that holds for a job, j_1, and a schedule, S, and states that if j_1 is an unassigned job in S, and all other jobs on which the assignment of j_1 depends are already assigned, then j_1 is an assignable job. The function **Relevant-Operators** take as an input a job that needs to be assigned and retrieve all the operators that are applicable to the selected jobs.

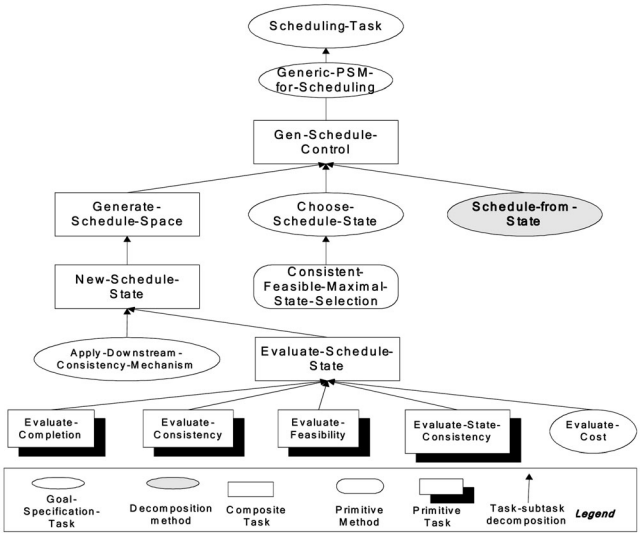


Fig. 2. The breakdown of the method independent control regime.

4.2 A Generic Problem Solving Model of Scheduling

Fig. 2 shows a breakdown of the method independent control regime of `Generic-Schedule`, whereby the scheduling task from the task ontology is decomposed into a number of (sub)tasks and (sub)methods are defined to achieve these tasks to construct a complete schedule.

The method independent control regime, `Gen-Schedule-Control` is a high-level control loop that takes as input a list of schedule operators and the scheduling task formalized in the scheduling task ontology (see Sections 3.1 and 3.2) and generates as output a complete schedule. The body of `Gen-Schedule-Control` first invokes the task `Generate-Schedule-Space`, which takes as input the scheduling task and returns either a schedule space, which consists of a number of schedule states or nothing. Then, the task `new-schedule-state` is invoked to create a root node. A root node is an initial schedule state which has an empty schedule associated with it because no jobs are assigned yet. Having generated a root node associated with a schedule space, the task `choose-schedule-state` is invoked to select an appropriate schedule state for expansion. Tasks `new-schedule-state` and `choose-schedule-state` are discussed below. Finally, the `schedule-from-state` is the last task invoked in the method independent control regime that takes as input the schedule state selected by `choose-schedule-state` and expands a schedule state by applying the relevant schedule operators. This task acts as a bridge between the method independent and the method specific control regime defined inside `schedule-from-state`.

4.2.1 Generation, Evaluation, and Selection of Schedule States

Task `new-schedule-state` creates a root node associated with a schedule space. In each newly created schedule state, we first apply the *downstream consistency enforcement* heuristic [59]. This heuristic propagates the earliest start time of a selected job downstream to check whether any other unassigned job has an earlier start time than the selected job. The complexity of this heuristic is linear and, in the absence of resource capacity conflict, it guarantees backtrack free search. Each newly generated schedule state is then evaluated by using the task `evaluate-schedule-state`. The following five methods are used to evaluate a schedule state (it is important to remember that these methods are independent of each other, i.e., a PSM that does not deal with cost issues will ignore a schedule state evaluation criterion that analyses cost)—`Evaluate-Completeness` checks whether a schedule associated with a state is a complete one. `Evaluate-Consistency` checks whether any of the constraints associated with a state are violated. `Evaluate-Feasibility` checks whether the requirements associated with a state are maintained. `Evaluate-Cost` calculates the costs of a state by using the cost function from the task ontology. `Evaluate-State-Consistency` is the most difficult task in the context of a state evaluation, which deals with checking whether a correct and consist state lay on a solution path. We used the full looking ahead and partial looking ahead heuristics [33]. The former heuristic checks the compatibility between any two unassigned jobs and the currently selected job and other assigned and unassigned jobs to ensure that the value requirements in terms of resources and time ranges of these jobs do not conflict with each other. The latter heuristic checks if the value requirements of any two unassigned jobs do not conflict with each other.

Having evaluated a schedule state, a scheduling agent has to select a correct schedule state which can be expanded to reach a solution. It is crucial to avoid a schedule state which is a nonsolution schedule state, i.e., a schedule state that violates constraints or requirements or a schedule state that already has a complete schedule associated with it. The task `choose-schedule-state` achieves the schedule state selection problem-solving action and the following methods are proposed to achieve this task—`Consistent-Maximal-Cheapest-State-Selection` selects a schedule state that does not violate constraints, provides maximal extension to a schedule, and has the least cost as compared to any other schedule states. `Consistent-Feasible-Maximal-State-Selection` selects a schedule state that does not violate constraints, maintains all the requirements, and provides maximal extension to a schedule (i.e., a maximum number of jobs can be assigned by selecting this schedule state). `Consistent-Cheapest-Maximal-State-Selection` selects a schedule state that does not violate any constraints, has the least cost, and provides maximal extension to a schedule. `Feasible-State-Selection` selects a schedule state that maintains all the requirements.

The main difference between these methods can be realized based on the order in which they consider the schedule state selection properties such as completion, constraint, or requirement violation. If no application specific knowledge is provided, then the method `Consistent-Feasible-Maximal-State-Selection` is used as a default schedule state selection strategy.

4.2.2 Method Specific Control

`Schedule-from-state` is a goal specification task, which takes as input a schedule state selected by the task `choose-schedule-state` and a schedule space, and then expands a selected schedule state iteratively until a solution schedule

is devised. This task is achieved by the default decomposition method `expand-incomplete-state`. This is one of the most important methods in `Generic-Schedule` because all the PSMs in our library are constructed by specializing this method. If a schedule state expanded by `expand-incomplete-state` is a complete one, then it is returned as a solution state, but, when an inconsistent or infeasible schedule state is encountered, then the search procedure generates a message to other tasks stating that a particular schedule state is a `deadend-state` and such a schedule state is marked as a nonsolution schedule state.

Schedule construction in `expand-incomplete-state` is achieved by using the notions of *context* and *focus* [46]. The notion of *context* specifies a problem-solving action that a PSM must execute for constructing a valid solution, e.g., the context in `Generic-Schedule` is to expand an incomplete schedule state to devise a complete schedule. The notion of *focus* identifies those variables that must be assigned to specific values to construct a solution, e.g., the main focus in `Generic-Schedule` is on one of the unassigned jobs that are assigned to resources and time ranges to construct a complete schedule.

`Generate-new-state-successor` is the main task invoked in the body of `expand-incomplete-state`. This task is decomposed into the following three subtasks: `resume-state`, `collect-state-foci`, and `propose-schedule-from-context`.

The task `resume-state` is invoked in a situation where a schedule state expansion cannot be finished due to say, constraint or the requirement violations, and needs to be resumed again once these violations are fixed. The task `collect-state-foci` is invoked in a problem-solving situation where a schedule state has not been extended yet. This task first collects all the foci (i.e., unassigned jobs) that can be assigned to resources and time ranges for constructing a complete schedule. Then, the task `propose-schedule-from-context` is invoked, which is a high-level control regime that takes as an input all the foci and then invokes the following tasks: `select-schedule-focus`, `collect-focus-operators`, `sort-focus-opera tors`, `generate-value-from-focus`, and `propose-schedule-from-focus` to assign all the jobs from the list of foci.

a. *Correct Job Selection: Select-Schedule-Focus*. The selection of a correct job is the most important task while constructing a schedule because it improves the efficiency of schedule construction by reducing unnecessary backtracking. Task `select-sche dule-focus` takes as input all the foci and selects a correct focus (i.e., a candidate job). We have defined the following eight different methods for judiciously selecting a correct job:

   1. `Job-Selection-Based-On-Lowest-De grees-Of-Freedom` subscribes to the *dynamic search rearrangement* (DSR) heuristic [14]. According to DSR, a job with the least number of resources and time ranges left for the assignment is selected as a candidate focus.
   2. `Job-Selection-Based-On-Due-Date` selects a job that has the earliest due date of unassigned jobs. Panwalkar and Iskander [51] list more than 100 job selection rules and one of the rules from their list selects a job based on its earliest due date. The main difference between their rule and ours is that a job in our heuristic is selected only when a selected job is competing with other jobs for the same resource, which has unary capacity and the time range of the selected job is overlapping with other jobs.
   3. `Job-Selection-Based-On-Start-Time` selects a job which has the earliest start time of unassigned jobs.
   4. `Job-Selection-Based-On-Precedence` sorts all the unassigned jobs based on the precedence relation among them and the first job from the sorted list is selected.
   5. `Job-Selection-Based-On-Minimal-Job-Dependency` subscribes to the minimal width ordering heuristic [25] according to which a highly constrained job is assigned first because it is deemed to reduce future backtracking.
   6. `Job-Selection-Based-On-Bottleneck-Resources` always gives priority to a job that consumes the bottleneck resources mainly because such a job is assumed to provide better control in maintaining the global stability of a schedule. The bottleneck resources are the ones whose individual capacity determines the overall productive capacity of a schedule.
   7. `Job-Selection-Based-On-Number-Of-Activities` selects a job that has the highest number of activities associated with it because such a job is believed to have more chances of conflicting with the resource and time range requirements of the other jobs.
   8. `Job-Selection-Based-On-Least-Num ber-Of-Activities` selects a job that has the least number of activities associated with it, particularly when the resource availability period of the resources required by a job are highly constrained.

   The method `Job-Selection-Based-On-Low est-Degrees-Of-Freedom` is used as a default focus selection method if no application specific knowledge is provided to select a candidate focus.

b. *Resource and Time Range Assignment*. Once a correct focus is selected, then the tasks `collect-focus-operators` and `sort-focus-operators` are invoked, first to collect and then to sort all the schedule operators that are applicable to assign the resources and time ranges to the selected focus. Finally, the tasks `generate-value-from-focus` and `propose-schedule-from-focus` are invoked. These two tasks take as an input the selected focus and sorted operators. The sorted operators are then applied to assign resources and a time range to the selected focus.

Once the assignment of a currently selected focus is completed then the task `new-schedule-state` is invoked again which repeats the complete problem-solving cycle until all the unassigned jobs in the list of collected foci are assigned to construct a complete schedule.

5 Engineering of Problem Solving Method from Our Library

Our library consists of seven PSMs: hill climbing, Propose & Backtrack (P&B) [58], Propose & Improve (P&I) [46], Propose & Revise (P&R) [40], [41], Propose & Restore-feasibility (P&Rf), Propose & Exchange (P&E) [53], and Propose and Genetical-Exchange. These PSMs cover and reason about all the types of schedule validation issues, such as completeness, constraint or requirement violation, and optimization. These methods are categorized according to the schedule validation issues handled by them—schedule completeness (`Generic-Schedule`, P&B), schedule optimization (hill climbing, P&I), schedule consistency and feasibility (P&R, P&E, P and GE, and P&Rf). The selection of a correct PSM is a key activity primarily when more than one PSM can be used to solve the application. For instance, both the P&R [40], [41] and P&E [53] methods can be used to fix constraint violations. The propose phase of P&R and P&E constructs a complete schedule and the main difference between these methods can be realized based on how they fix the constraint violations. The P&R method fixes the constraint violations by proposing a completely new set of assignments for the jobs involved in conflict, whereas the P&E method reorders the assignment of the jobs involved in conflict. The application specific knowledge requirements are taken into account for selecting these methods, i.e., if the constraint violation can be fixed only by proposing a completely new set of assignments, then the P&R method is used; otherwise, if the constraint violations can be fixed by reordering the assignments of the jobs involved in conflict, then the P&E method is used.

5.1 Schedule Modification Operators

The main aim of the operators (see Section 4.1.2) introduced in `Generic-Schedule` is to assign jobs to resources and time ranges, but they cannot deal with the constraint or requirement violations or the optimization issues. As a result, while engineering the PSMs in our library, we introduced `schedule-modification-resource-operator` and `schedule-modification-time-range-operator`. The former operator deals with the constraint or requirement violations that occur due to inconsistent resource assignments, while the latter operator deals with the constraint or requirement violations that occur due to conflicting time ranges assigned to jobs. Both the operators also deal with the schedule optimization issues.

5.2 Engineering of the Propose & Revise Method

Here, we describe the engineering of the Propose & Revise [41] method, which demonstrates how a new PSM in our library can be constructed simply by reusing and specializing `Generic-Schedule`. The engineering of all the PSMs in our library can be found in [56].

The P&R method [41] was originally developed to tackle the VT system for elevator configuration [40] and it was later extended to solve the production scheduling problem [42], [68]. Several researchers [19], [46], [75] have studied the P&R method; however, Motta [46] provides a much richer analysis of the P&R method in comparison with [19] and [75], where the P&R method is applied to solve parametric design problem by relating this to different constraint satisfaction techniques. Our aim here is also to provide a uniform support for constructing the P&R method by reusing `Generic-Schedule` and by teasing out the characteristics that are unique to scheduling.

5.2.1 Overview of the Propose & Revise Method

The propose phase of the P&R method constructs a complete schedule by assigning jobs to resources and time ranges. If any of the constraints are violated while constructing a schedule, then the revise phase of the method is invoked to fix these violations. The constraint violations are fixed by using the schedule modification operators (see Section 5.1). The notion of context and focus from `Generic-Schedule` is specialized according to the two phases associated with P&R. The context in the propose phase is to **extend** an incomplete schedule and the focus is on one of the **unassigned** jobs. The context in the revise phase is to **revise** an inconsistent schedule by fixing the constraint violations and the focus is on one of the **constraint violations**.

5.2.2 Method Specific Control Regime of P&R

The method independent control regime of the P&R method is similar to `Gen-Schedule-Control` (see Section 4.2) from `Generic-Schedule`.

The method specific control regime of P&R called, `propose-and-revise-control-structure` is constructed by specializing the method specific control regime of `Generic-Schedule` called, `expand-incomplete-state` (see Section 4.2.2). `Propose-and-revise-control-structure` first invokes the task `generate-new-state-successor` in the **extend** context to construct a complete schedule. If any of the constraints imposed on a schedule are violated while constructing a schedule, then they are fixed only when the complete schedule is constructed. The new task called `revise-schedule` associated with the method specific control regime of P&R is used to fix the constraint violations. The OCML definition of `revise-schedule` can be found in Fig. 3.

The task `revise-schedule` is achieved by using the following two alternative methods: `one-step-revision-for-constraint` and `fix-constraint-monotonically`.

The method `one-step-revision-for-constraint` can be used where only a single constraint is violated. This method takes as an input a schedule state violating a constraint and then it invokes the task `generate-new-state-successor` in the **revise** context. No special knowledge is required to select a focus (i.e., constraint violation) because only a single constraint violation can be fixed by using this method.

More interesting is the construction of the method `fix-constraint-monotonically`, which can be used to fix more than one constraint violation. It takes as an input a schedule state that has a number of constraint violations and then it invokes the task `generate-new-state-successor` in the **revise** context. The body of `fix-constraint-monotonically` invokes the task `collect-state-foci` in the **revise** context in order to collect all the constraint violations. This task is achieved by

```
(def-class REVISE-SCHEDULE (goal-specification-task) ?task
 ((has-input-role :value has-schedule-state)
  (has-output-role :value has-output-state)
  (has-schedule-state :type schedule-state)
  (has-output-state :type schedule-state)
  (has-goal-expression :value (kappa (?task ?s)
                                     (and (schedule-state ?s)
                                          (not (constraint-violations ?s ?any)))))))
```

Fig. 3. The OCML definition of `revise-schedule`.

```
(def-class SELECT-CANDIDATE-CONSTRAINT-VIOLATION (primitive-method) ?psm
 ((has-body :value (lambda (?psm)
                     (first (role-value ?psm 'has-schedule-foci)))))
 :own-slots ((tackles-task-type select-schedule-focus)
             (applicability-condition (kappa (?task)
                                             (= (role-value
                                                 ?task 'has-schedule-context)
                                             :revise)))))
```

Fig. 4. The OCML definition of `select-candidate-constraint-violation`.

defining the new method called `collect-all-constraint-violations`. Once the foci (i.e., constraint violations) are collected, then the candidate focus is selected according to the application-specific knowledge by invoking the task `select-schedule-focus` in the **revise** context and the new method, called `select-candidate-constraint-violation` is defined to achieve this task. The OCML definition of `select-candidate-constraint-violation` can be found in Fig. 4.

Once a correct focus is selected then the task `collect-focus-operators` is invoked to collect all the schedule modification operators that can be applied to fix the selected focus, and the new method called, `collection-of-applicable-fixes` is defined to achieve this task. The OCML definition of `collection-of-applicable-fixes` can be found in Fig. 5.

The task `sort-focus-operators` is used to sort the collected schedule modification operators. Finally, the first operator from the sorted list is used to fix the focus. The same problem-solving cycle is repeated until all the constraint violations from the list of foci are fixed.

6 Evaluation of the Library

Our library has been validated on five scheduling domains: satellite-scheduling, the CIPHER project schedule application, daily ship-maintenance, weekly ship-maintenance, and a benchmark application used in the scheduling area. For the sake of brevity, here, we will only discuss how the weekly ship-maintenance application is solved by using the PSMs in our library. A detailed discussion of the library evaluation on other applications can be found in [56].

6.1 The Weekly Ship-Maintenance Application

The weekly ship-maintenance is a real-life scheduling application. The primary aim of this application is to construct a *complete* and a *consistent* weekly schedule to perform different types of ship-maintenance activities. The working hours for each day are from 9:00 a.m. to 18:00 p.m.

6.1.1 Formalizing the Task Model of the Weekly Whip-Maintenance Application

This application consists of 21 ship-maintenance jobs which have to be assigned on 19 ship-maintenance resources. Each ship-maintenance job has a specific requirement for the resources on which they can be assigned and it also has a number of activities associated with it that must be executed to complete the ship-maintenance job. Each ship-maintenance job also has a time range within which all the activities must be accomplished. In order to formalize the ship-maintenance jobs, e.g., 12B3HTN, the new application specific class `12B3HTN-job` is defined. This class is defined as a subclass of the class `job` from the scheduling task ontology. As shown in the following box, the slots of the class `12B3HTN-job` are instantiated to represent the application specific knowledge associated with 12B3HTN. The formalization of the other ship-maintenance jobs and the activities and the time ranges associated with the jobs can be realized similarly. (See Fig. 6.)

Each ship-maintenance resource has a specific competence, which determines the specific types of ship-maintenance jobs it can handle for their execution. The ship-maintenance resources also have a fixed capacity, which determines the total number of ship-maintenance jobs they can handle at any given time. Finally, each ship-maintenance

```
(def-class COLLECTION-OF-APPLICABLE-FIXES (primitive-method) ?psm
 ((has-body :value (lambda (?psm)
                     (setofall ?op (and (schedule-modification-operator
                                         ?op applicable-to-constraints ?l)
                                        (member (role-value ?psm 'has-schedule-focus)
                                        (eval ?l)))))))
 :own-slots ((tackles-task-type '(collect-focus-operators))
             (applicability-condition
              (kappa (?task)
                     (and (= (role-value ?task has-schedule-context) :revise)
                          (constraint (role-value ?task has-schedule-focus)))))))
```

Fig. 5. The OCML definition of `collection-of-applicable-fixes`.

resource is available only during a restricted period. In order to formalize the application specific resources, e.g., ABF-BT-EN2, the new application specific class, called `ABF-BT-EN2-resource`, is defined as a subclass of the class `resource` from the scheduling task ontology. Then, the attributes of the class `ABF-BT-EN2-resource` is instantiated to represent the knowledge associated with ABF-BT-EN2. The formalization of the remaining resources can be understood similarly to ABF-BT-EN2.

6.1.2 Modeling Constraints and Requirements

The class `constraint` and `requirement` from the task ontology is instantiated to formalize the constraints and requirement specific knowledge, respectively. The weekly ship-maintenance application is formulated based on the following constraints and requirements elicited from application.

The **resource capacity constraint** is common to all the ship-maintenance resources. It states that a fixed capacity of the ship-maintenance resources, which determines the total number of ship-maintenance jobs each ship-maintenance resource can handle, must be maintained throughout schedule construction. A set of 19 resource capacity constraints are defined to impose this constraint on all the 19 ship-maintenance resources. The **daily frequency of ship maintenance job** constraint is common to all 21 ship-maintenance jobs. It states that all the ship-maintenance jobs must finish exactly within their daily working hours, 9:00 a.m. to 18:00 p.m. The **job working hour constraint** is again common to all the ship-maintenance jobs and states that, in the worse-case scenario, a ship-maintenance job may exceed its duration by not more than 10 minutes as long as this does not violate the daily frequency of a schedule. Finally, the **job priority requirement** states that if any two ship-maintenance jobs share the same ship-maintenance resource for their execution, then a weekly ship-maintenance job with higher duration gets priority.

```
(def-class 12B3HTN-JOB (job))

(def-instance 12B3HTN 12B3HTN-job
 ((has-activities turn-pump-shaft-by-hand)
  (requires-resource ABF-BT-EN2)
  (has-time-range 12B3HTN-time-range)))
```

Fig. 6. The OCML definition of the ship-maintenance job, 12B3HTN-JOB.

We did not encounter any particular problem while formalizing the task model of the weekly ship-maintenance application. Only a few application-specific relations and functions were defined to model constraints and requirements. More importantly, the classes like `job`, `resource`, `activity`, and `job-time-range` from the task ontology have provided a necessary level of detail to capture the application-specific knowledge precisely.

6.1.3 Construction of a Complete Schedule by Configuring the Propose & Backtrack Method

To solve this application, we first applied the P&B method [58] from our library mainly because as described in [58], the main property of this method is to construct a complete schedule by assigning jobs to resources and time ranges until an inconsistency is detected and then it backtracks to the last consistent schedule state, where different sets of resources and time ranges are tried to generate a consistent assignment. In other words, P&B always constructs a complete schedule, which matched with one of the goals of this application.

The P&B method was configured by defining the following two application specific operators, which were used to assign jobs to resources and time ranges—`weekly-ship-resource-operator` and `weekly-ship-time-range-operator`. These operators were defined uniformly as the subclasses of `schedule-extension-resource-operator` and `schedule-extension-resource-operator` (see Section 4.1.2), respectively. Finally, as shown in Fig. 7, the relation `schedule-operator-order` from `Generic-Schedule` was instantiated to determine the

```
(tell (schedule-operator-order 12B3HTN-JOB-to-ABF-BT-EN2-resource
                               12B3HTN-JOB-to-12B3HTN-time-range))
```

Fig. 7. The relation `schedule-operator-order` from `Generic-Schedule`.

order in which the operators were applied to assign a selected job, i.e., 12B3HTN.

According to the application specific knowledge, the ship-maintenance job with the least number of activities associated with it was selected as the candidate focus. The job selection method `job-selection-based-on-least-number-of-activities` (see Section 4.2.2, item a) from `Generic-Schedule` was successfully used to select the candidate focus.

The configuration of the P&B method was completed by determining the focus selection knowledge and no additional knowledge was required. The complete schedule for the weekly ship-maintenance application was constructed successfully by generating 1,245 schedule states. However, it was observed that the "daily frequency of ship-maintenance job" constraint imposed on the following four weekly ship-maintenance jobs: 482YTTN, 628URAN, 51A1BNN, and 266TFEN was violated and, therefore, the schedule was a complete, but it was not a consistent one. In order to fix the constraint violations, we decided to use the P&R method mainly because a completely new set of time ranges needed to be assigned to fix the constraint violations, and the reordering of their assignments failed to fix the violations.

6.1.4 Construction of a Complete and Consistent Schedule by Configuring the Propose & Revise Method

No knowledge was required to configure the propose phase of the P&R method. The revise phase was configured by defining the schedule modification operator, called `ship-maintenance-time-range-fix`, to fix the inconsistent time range assignment and it was defined as the subclass of `schedule-modification-time-range-operator` (see Section 5.1). This operator was defined in such a way that the weekly ship-maintenance jobs—482YTTN, 628URAN, 51A1BNN, and 266TFEN, shifted exactly by the same time (i.e., 10 minutes) by which they violated the "daily frequency of ship-maintenance job" constraint. In scheduling, this type of shift policy is referred to as the left-shift strategy [3], [4], [64]. The same aforementioned order of the ship-maintenance jobs was maintained to select the focus, i.e., constraint violation violated by them.

Once the configuration of the P&R method was completed, we again ran the weekly ship-maintenance application. After the completion of the revise phase, it was observed that our operators successfully shifted all the weekly ship-maintenance jobs by exactly the same time by which they violated the "daily frequency of ship maintenance job" constraint. The complete and consistent schedule for this application was constructed by generating 1,401 schedule states.

The evaluation study of our library shows that the PSMs in our library have generated a high number of schedule states while solving the applications. However, it is important to notice that very limited configuration efforts are required to solve these applications by using the PSMs in our library. And, only the application specific knowledge is used to configure these methods. Moreover, in comparison with more traditional approaches to schedule construction like constraint satisfaction, our library provides a much richer epistemological framework to analyze the various knowledge-intensive actions that occur in schedule construction.

7 Comparison with the Related Work

Here, we compare our library with some of the existing scheduling libraries—the production scheduling library [35], the constraint-satisfaction approach [38], the CommonKADS library [69], and, finally, the MULTIS-II library [72].

The major difference between our approach over that of Hori and Yoshida's [35] is that we subscribe to a top-down approach of schedule construction. It starts with the generic template (i.e., `Generic-Schedule`) whose components can be reused and refined by a configuration process to construct more specialized PSMs. As opposed to our approach, their library follows a bottom-up approach whereby all the problem-solvers are constructed by identifying and subscribing to the knowledge requirements of the production scheduling domain. Such a type of domain specificity restricts the possible reusability of their library to a single domain of scheduling. Another important difference between these two approaches is that, while the PSMs in their library can reason about only the completion and constraint violation issues of scheduling, our library provides a comprehensive repertoire of PSMs that tackles all the validation areas of scheduling. Moreover, the `Generic-Schedule` component of our library offers a much richer and quicker way to construct a new PSM simply by reusing its high-level tasks and by specializing the notions of *context*, *focus*, *state selection*, and *operator* construction knowledge. This uniformity allows us to compare and contrast the knowledge requirements of these PSMs. Because the `Generic-Schedule` component is absent in their library architecture, it does not offer a modularity for constructing a new PSM. From a scheduling perspective, their library discusses only two job selection criteria, i.e., earliest start time and down to the due-date, as compared to the broad job-selection criteria proposed in our library (see Section 4.2.2, item a).

The ILOG's [38] library subscribes to the constraint satisfaction (CS) approach as their problem-solving technique in contrast with the knowledge-intensive approach of our library. In spite of the uniform approach to modeling, CS fails to provide a fine-grained epistemological framework such as is required to analyze various knowledge-intensive tasks involved in the schedule construction process. It is essentially an implementation technique.

Because their library subscribes to CS, it aims at constructing sophisticated but domain-independent algorithms and such domain independence fails to support the important function of knowledge acquisition. Another primary difference between these two approaches is that ILOG focuses on the resource allocation class of the scheduling task as compared to the generic class of the scheduling task tackled by our library.

CommonKADS [6] is a comprehensive methodology which also tackles the assignment and scheduling tasks [69]. In their library, the problem-solvers are directly associated with the domain-specific knowledge which, in our view, makes it difficult to abstract the generic components associated with PSMs for their reuse. More importantly, the CommonKADS library is comprised of only one method, i.e., P&R. As a result, the CommonKADS library tackles only the completion and constraint violation issues of scheduling. In contrast with this, our library is comprised of seven different PSMs that allow us to tackle all the validation issues of scheduling. Moreover, the library framework of CommonKADS is opaque in nature as it fails to provide the required level of detail to construct a new PSM. In contrast with CommonKADS, our library provides a wide range of methods for selecting and evaluating a schedule state by considering different scenarios. Also, various job selection heuristics are provided that help to improve the efficiency of a schedule construction. Finally, our library offers a much richer framework to construct a new PSM simply by reusing the generic tasks developed in `Generic-Schedule` and by specializing the notions of *context*, *focus*, and the *state selection* policy.

The MULTIS-II library [72] also tackles the scheduling task at a generic level and, in this sense, is similar to our approach. However, some significant differences exist between the two approaches. Because a component like `Generic-Schedule` is absent in the MULTIS-II framework, this fails to abstract reusable tasks and methods from specialized PSMs. Therefore, the construction of new PSMs is very difficult in their framework. `Generic-Schedule` overcomes this problem by providing a clean separation between the method-specific and method independent components. While the PSMs in our library allow us to validate different types of schedules, a solution schedule in the MULTIS-II library is validated only against completion and constraint violation. From a scheduling perspective, `Generic-Schedule` provides a wide range of job selection methods to improve the efficiency of schedule construction. In contrast with our library, job selection in MULTIS-II is achieved entirely on the basis of *domain specific requirements*, which is not a very effective way to execute such an important problem-solving activity mainly because if wrong or partial domain knowledge is used to select a job, then the job selection component may end up selecting the wrong job, which could cause excessive backtracking.

8 Conclusion

In this paper, we have proposed a generic library of PSMs for the scheduling task. It is based on the TMDA knowledge modeling framework and follows a top-down approach. Because our library organization has drawn from the various KBS technologies, like ontologies, PSMs, search, and knowledge acquisition, it not only allows construction of different PSMs quickly, but also provides a way to compare and contrast their knowledge requirements. Our work is important for scheduling research both from theoretical and engineering perspectives. Theoretically, it exhibits a nice integration of the various techniques that have been developed in scheduling research and also provides an insight into the various components which can be used in scheduling. From the engineering perspective, our library offers support for the rapid construction of scheduling applications from different domains. Moreover, our library provides a comprehensive repertoire of seven different PSMs, which allows us to reason about all the schedule validation issues, such as completion, constraint and requirement violation, and optimization. Finally, our library now has hundreds of reusable definitions and it has been validated on a number of real-life and benchmark applications to confirm its generic nature.

Acknowledgments

The authors would like to thank Victoria Uren for reading earlier drafts of this paper. They would also like to thank the anonymous reviewers for their helpful discussion and providing comments that helped to improve the quality of this paper. This research was partially supported by the Advanced Knowledge Technologies (AKT) project which is sponsored by the UK Engineering and Physical Sciences Council under grant number GR/N15764/01.

References

[1] J.F. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. ACM*, vol. 26, no. 11, pp. 832-843, 1983.

[2] J. Angele, D. Fensel, D. Landes, and R. Studer, "Developing Knowledge-Based System with MIKE," *Automated Software Eng.*, vol. 5, no. 4, pp. 389-418, 1998.

[3] K.R. Baker, *Introduction to Sequencing and Scheduling.* John Wiley and Sons, 1974.

[4] J.C. Beck and M.S. Fox, "A Generic Framework for Constraint-Directed Search and Scheduling," *AI Magazine*, vol. 19, no. 4, pp. 101-130, 1998.

[5] V.R. Benjamins, "Problem Solving Methods for Diagnosis," PhD dissertation, Dept. of Social Sciences Informatics, Univ. of Amsterdam, The Netherlands, 1993.

[6] J. Breuker and W. Van de Velde, *CommonKADS Library for Expertise Modelling.* Amsterdam: IOS Press, 1994.

[7] V. Brusoni, L. Console, E. Lamma, P. Mello, M. Milano, and P. Terenziani, "Resource-Based vs. Task-Based Approaches for Scheduling Problems," *Proc. Ninth Foundations of Intelligent Systems (ISMIS '96)*, pp. 325-334, 1996.

[8] P. Burke, "Scheduling in Dynamic Environments," PhD dissertation, Univ. of Strathclyde, U.K., 1989.

[9] P. Burke and P. Prosser, "A Distributed Asynchronous Scheduler," *Intelligent Scheduling*, M. Zweben and M.S. Fox, eds., chapter 11, pp. 309-339, San Francisco: Morgan Kauffman, 1994.

[10] B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, vol. 1, pp. 23-30, 1986.

[11] B. Chandrasekeran and S. Mittal, "Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving," *Int'l J. Human-Computer Studies*, vol. 51, no. 2, pp. 357-368, 1999.

[12] E. Coelho and G. Lapalme, "Describing Reusable Problem-Solving Methods with a Method Ontology," *Proc. 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, B.R. Gaines and M. Musen, eds., pp. 1-20, 1996.

[13] A. Collinot, C. Le Pape, and G. Pinoteau, "SONIA: A Knowledge-Based Scheduling System," *Artificial Intelligence in Eng.,* vol. 3, no. 2, pp. 86-94, 1988.

[14] R. Dechter and I. Meiri, "Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems," *Proc. 11th Int'l Joint Conf. Artificial Intelligence (IJCAI '89),* pp. 271-277, 1989.

[15] J. Dorn and W. Slany, "A Flow Shop Compatibility Constraints in a Steel Manufacturing Plant," *Intelligent Scheduling,* M. Zweben and M.S. Fox, eds., chapter 22, pp. 629-654, San Francisco: Morgan Kauffman, 1994.

[16] J. Domingue, E. Motta, and S. Watt, "The Emerging VITAL Workbench," *Proc. Seventh European Workshop Knowledge Acquisition for Knowledge-Based Systems,* N. Aussenac, G. Boy, B.R. Gaines, M. Linster, J.-G. Ganascia, and Y. Kodratoff, eds., pp. 320-339, 1993.

[17] A. Farquhar, R. Fikes, and J. Rice, "The Ontolingua Server: A Tool for Collaborative Ontology Construction," *Int'l J. Human-Computer Studies,* vol. 46, no. 6, pp. 707-728, 1997.

[18] D. Fensel, "Assumptions and Limitations of Problem-Solving Method: A Case Study," *Proc. Ninth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* B. Gaines and M. Musen, eds., 1995.

[19] D. Fensel and R. Straatman, "The Essence of Problem-Solving Methods: Making Assumptions to Gain Efficiency," *Int'l J. Human-Computer Studies,* vol. 48, pp. 181-215, 1998.

[20] D. Fensel and E. Motta, "Structured Development of Problem Solving Methods," *IEEE Trans. Knowledge and Data Eng.,* vol. 13, no. 6, pp. 913-932, Nov./Dec. 2001.

[21] M.S. Fox, "An Organisation View of Distributed Systems," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 11, pp. 70-80, 1981.

[22] M.S. Fox and S. Smith, "ISIS: A Knowledge-Based System for Factory Scheduling," *Int'l J. Expert Systems,* vol. 1, no. 1, pp. 25-49, 1984.

[23] M.S. Fox and N. Sadeh, "Why Scheduling is Difficult? A CSP Perspective," *Proc. Ninth European Conf. Artificial Intelligence (ECAI '90),* pp. 754-767, 1990.

[24] B.R. Fox and K.G. Kempf, "Planning, Scheduling, and Uncertainty in the Sequence of Future Events," *Proc. Second Ann. Conf. Uncertainty in Artificial Intelligence (UAI '86),* pp. 395-402, 1986.

[25] E.C. Freuder, "A Sufficient Condition for Backtrack-Free Search," *J. ACM,* vol. 29, no. 1, pp. 24-32, 1982.

[26] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Co., 1979.

[27] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research,* vol. 13, no. 5, pp. 533-549, 1986.

[28] T.J. Grant, "Lessons for OR from AI: A Scheduling Case Study," *J. Operations Research Soc.,* vol. 37, pp. 41-57, 1986.

[29] T.R. Gruber, "A Translation Approach to Portable Ontologies," *Knowledge Acquisition,* vol. 5, no. 2, pp. 199-220, 1993.

[30] A. Gomez-Perez and V.R. Benjamins, "Applications of Ontologies and Problem Solving Methods," *AI Magazine,* vol. 20, no. 1, pp. 119-122, 1999.

[31] N. Guarino, "Understanding, Building and Using Ontologies," *Int'l J. Human-Computer Studies,* vol. 46, nos. 2-3, pp. 293-310, 1997.

[32] K. Hadavi, W-L. Hsu, T. Chen, and C-N. Lee, "An Architecture for Real-Time Distributed Scheduling," *AI Magazine,* vol. 13, no. 3, pp. 46-56, 1992.

[33] R.M. Haralick and G.L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence,* vol. 14, pp. 263-313, 1980.

[34] M. Hori, Y. Nakamura, H. Satoh, K. Maruyama, T. Hama, S. Honda, T. Takenaka, and F. Sekine, "Knowledge-Level Analysis for Eliciting Composable Scheduling Knowledge," *Artificial Intelligence in Eng.,* vol. 9, no. 4, pp. 253-264, 1995.

[35] M. Hori and T. Yoshida, "Domain-Oriented Library of Scheduling Methods: Design Principle and Real-Life Application," *Int'l J. Human-Computer Studies,* vol. 49, pp. 601-626, 1998.

[36] K. Kempf, C. Le Pape, S.F. Smith, and B.R. Fox, "Issues in the Design of AI-Based Schedulers: A Workshop Report," *AI Magazine,* vol. 11, no. 5, pp. 37-46, 1991.

[37] C. Kruger, "Software Reuse," *Computing Surveys,* vol. 24, no. 2, pp. 131-183, 1992.

[38] C. Le Pape, "Implementation of Resource Constraints in ILOG SCHEDLE: A Library for the Development of Constraint-Based Scheduling Systems," *Int'l Systems Eng.,* vol. 3, no. 2, pp. 55-66, 1994.

[39] S. Marcus, *Automatic Knowledge Acquisition for Expert Systems.* Kluwer Academic, 1988.

[40] S. Marcus, J. Stout, and J. McDermott, "VT: An Expert Elevator Designer that Uses Knowledge-Based Backtracking," *AI Magazine,* vol. 9, no. 1, pp. 95-111, 1988.

[41] S. Marcus and J. McDermott, "SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems," *Artificial Intelligence,* vol. 39, no. 1, pp. 1-37, 1989.

[42] J. McDermott, "Towards a Taxonomy of Problem Solving Methods," *Automating Knowledge Acquisition for Expert Systems,* S. Marcus, ed. Kluwer Academic, 1988.

[43] D.L. McGuinness and F. van Hermelen, "OWL Web Ontology Language Overview," *W3C Recommendation,* http://www.w3c.org/TR/2004/REC-owl-guide-20040210/, Feb. 2004.

[44] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda, "Task Ontology for Reusable Problem Solving Knowledge," *Towards Very Large Knowledge Bases,* pp. 46-57, IOS Press, 1995.

[45] D.H. Mott, J. Cunningham, G. Kelleher, and J.A. Gadsden, "Constraint-Based Reasoning for Generating Naval Flying Programmes," *Expert Systems,* vol. 3, no. 2, pp. 226-246, 1988.

[46] E. Motta, *Reusable Components for Knowledge Modelling: Principles and Case Studies in Parametric Design.* Amsterdam: IOS Press, 1999.

[47] E. Motta, D. Rajpathak, Z. Zdrahal, and R. Roy, "The Epistemology of Scheduling Problems," *Proc. 15th European Conf. Artificial Intelligence (ECAI '02),* F. van Harmelen, ed., pp. 215-219, 2002.

[48] M. Musen, S.W. Tu, H. Eriksson, J.H. Gennari, and A.R. Puerta, "PROTÉGÉ-II: An Environment for Reusable Problem-Solving Methods and Domain Ontologies," *Proc. 13th Int'l Joint Conf. Artificial Intelligence (IJCAI '93),* 1993.

[49] R. Neches, R.E. Fikes, T. Finin, T.R. Gruber, T. Senator, and W.R. Swartout, "Enabling Technology for Knowledge Sharing," *AI Magazine,* vol. 12, no. 3, pp. 36-56, 1991.

[50] A. Newell and H. Simon, "Computer Science as Empirical Enquiry: Symbols and Search," *Comm. ACM,* vol. 19, no. 3, pp. 113-126, 1976.

[51] S.S. Panwalkar and W. Iskander, "A Survey of Scheduling Rules," *Operations Research,* vol. 25, no. 1, pp. 45-61, 1977.

[52] H. Parunak, B. Irish, J. Kindrick, and P. Lozo, "Fractal Actors for Distributed Manufacturing Control," *Proc. Second IEEE Conf. Artificial Intelligence Applications,* pp. 653-660, 1985.

[53] K. Poeck and F. Puppe, "COKE: Efficient Solving of Complex Assignment Problems with the Propose-and-Exchange Method," *Proc. Fifth Int'l Conf. Tools with Artificial Intelligence,* pp. 136-143, 1992.

[54] P. Prosser and I. Buchanan, "Intelligent Scheduling: Past, Present, and Future," *Int'l Systems Eng.,* vol. 3, no. 2, pp. 67-78, 1994.

[55] D. Rajpathak, E. Motta, Z. Zdrahal, and R. Roy, "A Generic Library of Problem Solving Methods for Scheduling Applications," *Proc. Second Int'l Conf. Knowledge Capture (K-CAP '03),* pp. 113-120, 2003.

[56] D. Rajpathak, "A Generic Library of Problem Solving Methods for Scheduling Applications," PhD dissertation, Knowledge Media Inst., The Open Univ., U.K., 2004.

[57] A. Ravidran, D.T. Philips, and J.J. Solberg, *Operations Research Principles and Practice.* John Wiley and Sons Inc., 1987.

[58] T. Runkel, W.B. Birmingham, and A. Balkany, "Solving VT by Reuse," *Int'l J. Human-Computer Studies,* vol. 44, nos. 3/4, pp. 403-433, 1996.

[59] N. Sadeh, "Micro-Opportunistic Scheduling: The Micro-Boss Factory Scheduler," *Intelligent Scheduling,* M. Zweben and M.S. Fox, eds, chapter 4, pp. 99-135, San Fracisco: Morgan Kauffman, 1994.

[60] J. Saucer, "Knowledge-Based Design of Scheduling Systems," *Int'l J. Intelligent Automation and Soft Computing,* vol. 7, no. 1, pp. 55-62, 1997.

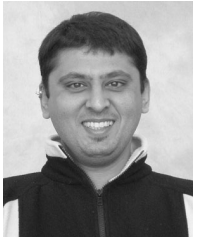
[61] G. Schreiber, B. Wielinga, R. De Hogg, H. Akkermans, and W. Van de Velde, "CommonKADS: A Comprehensive Methodology for KBS Development," *IEEE Expert,* vol. 9, no. 6, pp. 28-37, 1999.

[62] S.D. Sharma, *Operations Research.* Meerut, India: Kedarnath-Ramnath and Co., 1996.

[63] J.R. Slagle and H. Hamburger, "An Expert System for a Resource Allocation Problem," *Comm. ACM,* vol. 28, no. 9, pp. 994-1004, 1985.

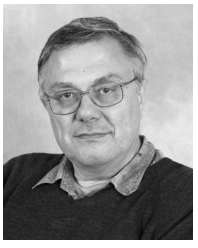
[64] S. Smith, P.S. Ow, N. Muscettola, J.Y. Potvin, and D. Matthys, "OPIS: An Opportunistic Factory Scheduling System," *Proc. Third Int'l Conf. Industrial and Eng. Appications of Artificial Intelligence and Expert Systems,* May 1990.

[65] S.F. Smith, "Reactive Scheduling Systems," *Intelligent Scheduling Systems,* D.E. Brown and W.T. Scherer, eds., Kluwer Press, 1995.
[66] S. Smith and M.A. Becker, "An Ontology for Constructing Scheduling Systems," *Working Notes of AAAI Symp. Ontological Eng.,* 1997.
[67] L. Steels, "Components of Expertise," *AI Magazine,* vol. 11, pp. 28-49, 1990.
[68] J. Stout, G. Caplain, S. Marcus, and J. McDermott, "Towards Automating Recognition of Differing Problem-Solving Demands," *Int'l. J. Man Machine Studies,* vol. 29, no. 5, pp. 599-611, 1988.
[69] U. Sundin, "Assignment and Scheduling," *CommonKADS Library for Expertise Modelling,* J. Breuker and W. Van de Velde, eds. Amsterdam: IOS Press, pp. 231-263, 1994.
[70] B. Swartout and Y. Gil, "EXPECT: Explicit Representations for Flexible Acquisition," *Proc. Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop,* 1995.
[71] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadbolt, "Knowledge Acquisition Support through Generalised Directive Models," *Second Generation Expert Systems,* J.M. David, J.P. Krivine, and R. Simmons, eds. Springer-Verlag, pp. 428-455, 1993.
[72] Y.A. Tijerino and R. Mizoguchi, "MULTIS II: Enabling End-Users to Design Problem-Solving Engines via Two-Level Task Ontologies," *Proc. Seventh European Workshop Knowledge Acquisition for Knowledge-Based Systems (EKAW '93),* N. Aussenac-Gilles, G.A. Boy, B.R. Gaines, J. Ganascia, Y. Kodratoff, and M. Linster, eds., 1993.
[73] A. Valente, V.R. Benjamins, and L. Nunes de Barross, "A Library of System-Derived Problem-Solving Methods for Planning," *Int'l J. Human-Computer Studies,* vol. 48, no. 4, pp. 417-447, 1998.
[74] G. van Heijst, P. Terpstra, B.J. Wielinga, and N. Shadbolt, "Using Generalised Directive Models in Knowledge Acquisition," *Proc. Sixth European Knowledge Acquisition Workshop,* pp. 112-132, 1992.
[75] B. Wielinga, J.M. Akkermans, and A.T. Schreiber, "A Formal Analysis of Parametric Design Problem Solving," *Proc. Ninth Banff Knowledge Acquisition Workshop for Knowledge-Based System (KAW '94),* B.R. Gaines and M.A. Musen, eds. 1995.



Dnyanesh G. Rajpathak received the first degree in production engineering from the University of Pune, the master's degree in advanced manufacturing management and technology from the University of Surrey, and the PhD degree in artificial intelligence from the Open University. He is a research fellow at the Knowledge Media Institute. He has been conducting research for the last five years in the areas of scheduling, knowledge engineering, ontology engineering, and problem-solving methods. His current research interests include planning, context modeling, ontology integration and alignment, primarily for the semantic Web technology.

Enrico Motta received the first degree in computer science from the University of Pisa in Italy and the PhD degree in artificial intelligence from the Open University. He is a professor and director of the Knowledge Media Institute (KMi) at The Open University. His current research focuses primarily on the integration of semantic, Web and language technologies to support the development of intelligent web applications. Over the years, he has led KMi's contribution to numerous high-profile projects, such as the highly prestigious, EPSRC-funded Interdisciplinary Research Collaboration on Advanced Knowledge Technologies (AKT), as well as several EU-funded ones, most recently NeOn, X-Media, and Open Knowledge. He is editor-in-chief of the *International Journal of Human Computer Studies* and is also on the editorial board of *IEEE Intelligent Systems* and the *Journal of Web Semantics*. He is also director of the European Summer School on Ontological Engineering and the semantic Web, which is now in its third edition. He is the author of more than 120 refereed publications and a book, *Reusable Components for Knowledge Modelling*, published by IOS Press. He chaired the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004) and was the program chair of the Fourth International Semantic Web Conference (ISWC 2005).



Zdenek Zdrahal is a senior research fellow at the Knowledge Media Institute of The Open University. He has been conducting research in the areas of knowledge engineering, knowledge management, problem-solving methods, knowledge-based system life cycle support, and organizational learning.

Rajkumar Roy is a professor at Cranfield University and has a background in manufacturing engineering and artificial intelligence. He is working in the areas of knowledge engineering, decision support, and shop floor implementation of expert systems. He is currently leading the research in the decision engineering area at Cranfield. The research theme includes engineering cost estimating, design optimization, and micro-knowledge management.